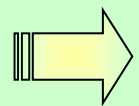


高级语言程序设计

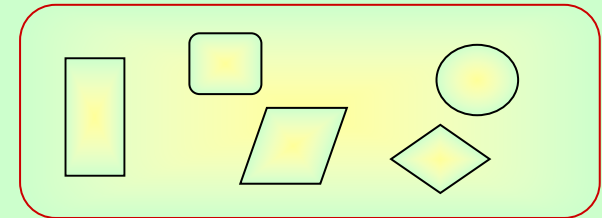
第 6 章 数组

华中师范大学物理学院 李安邦

- 程序的活动都与数据有关：输入/输出、存储、计算等。
- 需要处理的信息多种多样。数据可能简单或复杂，数据间可有丰富多采、密切或松弛的关系。
- 语言需要提供一套[数据机制](#)，描述与数据有关的问题：必须足够丰富，以满足需要；不能过庞杂，臃肿难用；也不能太低级，使描述过于烦琐。
- 高级语言的通行方式：
 - 把数据分为类型，每个类型是一个数据集。
 - 一组基本数据类型；书写方式；基本操作。
 - 一组构造复合数据类型或对象的机制。



- 组合的数据对象称为复合数据对象。
- 复合对象形成的类型称为复合数据类型，组成部分称为成分/成员/元素。



- 可创建能存放复合类型数据的变量。它们可作为整体使用，通过名字可以访问整个复合对象。
- 提供访问复合数据对象成分的操作，以存取复合变量的成分：使用成分的值或给成分赋值。
- 常见组合机制是数组和结构体，另一重要机制是指针，用于构造更复杂的数据结构。

第6章 数组

6.1 数组的概念、定义和使用

6.1.1 数组变量定义

6.1.2 数组的使用

6.1.3 数组的初始化

6.2 数组程序实例

6.3 数组作为函数参数

6.4 二维和 multidimensional 数组

6.5 字符数组与字符串

6.6 编程实例

6.1 数组的概念、定义和使用

数组 (array) 是多个同类型数据对象的组合。

一个数组汇集了多个数据项，数组元素。

可从数组出发处理各元素，以统一方式处理一批/所有元素，是数组和一组独立变量的主要区别。

为此需要：

- 数组**描述**，数组变量定义
- 数组**使用**，包括通过数组变量使用其元素
- 数组**实现**，数组的存储方式

6.1.1 数组变量的定义

定义数组变量（简称“数组”）时需要说明：

- 数组元素类型，变量名
- 数组（变量）的元素个数（数组大小或长度）



用方括号内的整型表达式说明元素个数，按照ANSI C的要求，表达式应该能够静态地确定其值。

例：定义两个数组：

```
int array[8];  
double db[20];
```

元素类型 变量名 元素个数

可用 **const** 关键词先定义整型常变量或用 **enum** 定义枚举常量，然后用这些常量来指定数组大小；或者用 **#define** 做一个简单宏定义，然后用宏名来指定数组大小。

可以在一条语句中定义多个数组变量。

```
const int NUM = 10;
int a1[NUM], a2[NUM + 1];
double db1[NUM], db2[NUM + 2];

enum {LEN = 20};
int a3[LEN], a4[LEN * 2];
double db3[LEN], db4[LEN * 3];

#define SIZE 100
int a5[SIZE], a6[SIZE * 4 + 1];
double db5[SIZE], db6[SIZE * 5 - 2];
```

按照 ANSI C 标准，在定义数组时**不能用无法在编译时静态求值的**整型变量或表达式来指定数组大小。

例如下面这个定义方式是不合法的（可以通过编译，但是在运行时会出错）：

```
int n;  
int arr[n];
```

下面函数中的数组定义按照 ANSI C 标准也是不合法的。

```
void f(int m, int n) {  
    int b[n];  
    ....  
}
```

↗
局部数组 b 的大小依赖于函数的参数
值，该值在编译时无法确定。

（新 C99 标准支持这种定义）

数组定义可以与其它变量的定义混和写在一起：

```
int m, n, a2[16], a3[25];  
double db2[20], db3[50], x, y;
```

数组变量定义可以出现在任何能定义简单变量的地方，而且在作用域和存在期方面与简单变量没有差别。

根据定义位置不同，数组分为**外部数组**和函数内的**局部数组**，包括函数内的静态局部数组和普通的自动数组。

定义方式（及位置）决定了它们的作用域与存在期。

在大型程序中有时还需要写出数组的**外部声明**。

此时不必写数组大小，只要在数组变量名后写一对方括号：

```
extern int a1[];  
extern double db1[];
```

复习：

- **局部变量（自动变量）**：定义在函数内部，前面不加 static 。由于它们的作用域是限制在所属的复合结构内部，所以也叫**局部变量**。因为它们的存储空间的创建和销毁都是由系统自动处理的，所以叫**自动变量**。（自动局部变量）
- **外部变量（全局变量）**：定义在函数外部的变量。由于作用域是全局性的，所以也叫全局变量。全程存在期，一次初始化。
- **静态局部变量**：定义在函数内部，前面加 static 。局部作用域，全程存在期，一次初始化。

6.1.2 数组使用

元素顺序编号，首元素序号 0，其余顺序编号。

长度为 LEN 的数组的元素编号是 0 到 LEN-1。



定义：int a[8];

元素编号为 0、1、2、...、7。称为下标或指标。

元素访问通过 [] 运算符，优先级最高，运算对象是数组名和括号里表示下标的表达式。

表达式或语句里的“a[0]”、“a[3]”、“a[6]”这种写法称为下标表达式。

例：有上面定义后，可写：

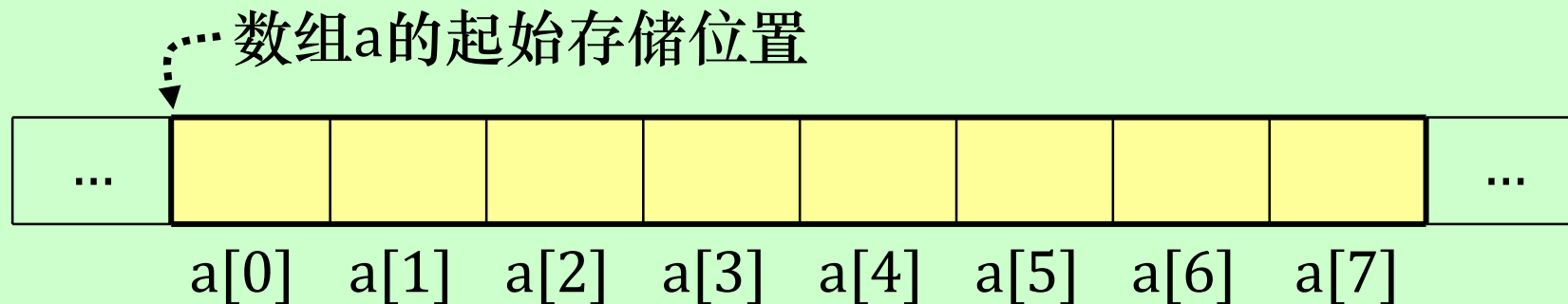
```
a[0] = 1; a[1] = 1;  
a[2] = a[0] + a[1];
```

数组的存储实现

数组占据一片连续存储区，元素顺序排列，0号元素在最前面，各元素占相同空间。如有：

```
int a[8];
```

a 的存储恰好能存放 8 个整型数据：



a 至少相当于 8 个 int 变量，a[0] ~ a[7] 可以看作变量名。保证元素排在一起，能以统一方式使用。

```
a[0] = 1; a[1] = 1; a[2] = a[0] + a[1];
```

简单情况可用简单变量代替数组，如定义: int b0, b1, b2, b3;
可以写类似语句。这没有表现出数组的价值。

数组意义是能以统一方式描述对一组数据的处理。

下标表达式可用一般的整型表达式。如：



```
a[i] = a[i-1] + a[i-2];
```

The expression `a[i-1]` and `a[i-2]` are crossed out with red lines. Two red arrows point from the `i` in `a[i]` to the `i-1` and `i-2` in the previous terms, illustrating how the value of `i` determines which elements are accessed.

访问哪个元素由 i 值确定。同一语句可以访问不同元素，可用在循环里访问一批元素。

```
for (i = 0; i < 100; ++i) {  
    b[i] += a1[i] * a2[i];
```

```
} // 假设已定义数组 b[100]、a1[100]和a2[100]  
循环中涉及到300个基本数据对象。
```

【例6-1】 写程序创建包含前30个 Fibonacci 数的数组，然后从小到大打印数组中所有的数。

```
int main() {  
    const int NUM = 30;  
    long fib[NUM];  
    int n;  
    fib[0] = 1;  fib[1] = 1;  
    for (n = 2; n < NUM; ++n) //计算  
        fib[n] = fib[n-1] + fib[n-2];  
    cout << "Fibonacci sequence:" << endl;  
    for (n = 0; n <= NUM-1; ++n) //打印输出  
        cout << fib[n] << (n % 5 == 4 ? '\n' : '\t');  
    return 0;  
}
```

对数组的多个或全部元素操作，常用 for 语句。

令变量遍历数组下标：

```
for (n = 0; n < 数组长度; ++n) ...
```

```
for (n = 0; n <= 数组长度 - 1; ++n) ...
```

n 取值与数组下标相同。

问题：fib 是 NUM 个元素的数组，假设程序里写：

```
for(n = 2; n <= NUM; ++n)
```

```
    fib[n] = fib[n-1] + fib[n-2];
```

循环中试图访问 fib[**NUM**]，实际无此元素。

用超范围的下标访问称为越界访问，是数组使用中最常见的错误。下标值超范围是运行中的问题。

C/C++ 不检查数组元素访问的合法性，运行中出现越界不会报错。

超范围访问是严重错误，后果无法预料。

可能的后果：

- 某些系统（如Windows）中，越界访问可能导致动态错，系统会强行终止出错的程序。
- 某些系统（如DOS）不检查非法访问，越界可能破坏本程序的数据/程序本身/其他软件，甚至操作系统。

编程者要保证数组下标值的合法性，保证不越界。

6.1.3 数组初始化

定义数组时可直接初始化。外部数组、自动数组和静态局部数组都可在定义时进行初始化。

写法：= {初值表达式, 初值表达式, ...}

定义时初始化的例子：

```
int b[4] = {1, 1, 2, 3};
```

```
double ax[6] = {1.3, 2.24, 5.11, 8.37, 6.5};
```

初值表达式必须是常量表达式。

这种写法只能用于数组初始化，不能用在语句里。

初始化语句的特殊用法:

1、只为部分元素提供初值，其余元素将自动赋初值 0。初始化的元素个数不得超过数组元素个数。

例: `long fib[NUM] = {1, 1};`

数组中其它元素将自动赋初值 0。

常见方式: 把首元素赋初值 0 (其它元素就会自动赋初值 0) : `int a[NUM] = {0};`

2、若给了所有元素的初值，可以不写数组大小而只写方括号，元素个数由初值个数确定。

例: `int a[] = {1,1,2,3,5,8,13,21,34,55};`

这种写法能减少维护负担，有利于程序修改。

求数组长度: `sizeof(a)/sizeof(a[0])`

若定义时未初始化，外部数组和静态局部数组的元素自动初始化为 0；自动数组不自动初始化。

【例6-2】写一个程序，里面包含有未初始化的外部数据和函数内的自动数组，对比它们的初值。

```
#include <iostream>
using namespace std;
const int NUM = 8; //定义全局整型常变量NUM
int ga[NUM]; //定义外部数组ga，未初始化
int main() {
    int array[NUM]; //局部数组array，未初始化
    for (int i = 0; i < NUM; i++)
        cout << ga[i] << (i == NUM-1 ? "\n" : "\t");
    for (int i = 0; i < NUM; i++)
        cout << array[i] << (i == NUM-1 ? "\n" : "\t");
    return 0;
}
```

0	0	0	0	0	0	0	0
??	??	??	??	??	??	??	??

6.1.4 C99 和C++ 中的变长数组

C99 和C++ 中允许定义**变长数组**：长度不是在编译时确定，而是在运行时确定，在不同运行时的长度可能不同。

(1) 使用已有合法值的整型变量作为数组长度

```
int n = 10;  
int array1[n];  
cin >> n;  
int array2[n];
```

存在安全隐患，需要
谨慎处理！

(2) 使用参数值作为数组长度

```
void func (int m) {  
    int b1[m], b2[m+10];  
    ...  
}
```

建议初学者不用。
(如果要用，就必须
学习完整用法)

第6章 数组

6.1 数组的概念、定义和使用

6.2 数组程序实例

6.2.1 从字符到下标（整数）

6.2.2 筛法求质数

6.2.3 约瑟夫问题

6.2.4 多项式求值

6.2.5 定义数组的考虑

6.3 数组作为函数参数

6.4 二维和 multidimensional 数组

6.5 字符数组与字符串

6.6 编程实例

示例程序表现了数组编程的一些常用方法和技巧，其中一些方法有广泛意义。

6.2.1 从字符到下标（整数）

【例6-3】写程序统计由标准输入得到的数字字符的个数。
用字符分类函数 `isdigit` 判断是否数字字符，然后计数。

不用数组：定义10个计数变量，都赋初值0，用 `if` 或 `switch` 区分情况，遇数字字符（`isdigit`）时对应计数器加1。

```
int a0=0, a1=0, a2=0, a3=0, a4=0, a5=0, a6=0, a7=0, a8=0,
a9=0;
```

```
while ((ch = getchar()) != EOF)
    if (isdigit(ch)) { //是数字字符
        if (ch=='0') a0++;
        else if (ch=='1') a1++;
        else if (ch=='2') a2++;
        else if (ch=='3') a3++;
        .....
    }
}
```

用数组的办法：定义数组 cs[10]，用于计数。

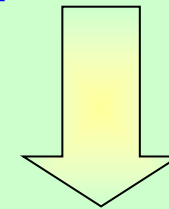
int cs[10] = {0}; // 定义数组，用于计数

```
while ((cin >> ch) != EOF){  
    if (ch=='0') cs[0]++;  
    else if (ch=='1') cs[1]++;  
    else if (ch == '2') cs[2]++;  
    else if (ch == '3') cs[3]++;  
    else if (ch == '4') cs[4]++;  
    else if (ch == '5') cs[5]++;  
    else if (ch == '6') cs[6]++;  
    else if (ch == '7') cs[7]++;  
    else if (ch == '8') cs[8]++;  
    else if (ch == '9') cs[9]++;  
}
```

数字字符顺序排列

int k = ch - '0';

cs[k]++;



cs[ch-'0']++ ;

从字符到下标

西文字符 ASCII 码 (美国信息交换标准代码)

ASCII 值	字符	ASCII 值	字符	ASCII 值	字符	ASCII 值	字符
0	NUL	32	(space)	64	④	96	,
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	,	71	G	103	g
8	BS	40	(72	H	104	h
9	HT	41)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	■	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	X	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	/	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	—	127	DEL

控制符

编码为 33 - 126
的字符为可打印
的人读字符

控制符


```
int main () {  
    int ch, i, cs[10] = {0};  
    cout << "input some chars (Ctrl+Z to end) : ";  
    while ((ch = getchar()) != EOF)  
        if ( isdigit(ch) )  
            cs[ch - '0']++;  
    for (i = 0; i < 10; ++i)  
        cout << "Number of " << i << ": " << cs[i] << endl;  
    return 0;  
}
```

6.2.2 筛法求质数

【例6-4】筛法是求质数的著名方法。具体算法：

取 2 开始的整数序列：

1. 令 n 等于 2，它是质数；
2. 划掉序列中所有 n 的倍数；
3. 令 n 等于下一未划元素(是质数)，回到步骤 2。

用数组表示整数序列：定义数组 an


以元素下标表示对应整数： $an[i]$ 的下标 i 代表相应整数 i 。

用元素的值表示未划掉 (1) 或划掉 (0)。

➔ 开始时元素置 1，而后不断将元素置 0，直到确定了给定范围里的所有质数。

假设NUM是给定范围：

```
//定义数组an，元素初始化1，  
//将an[0]和an[1]置0（它们不是质数）  
for (int i = 2; i 值不大于某个数; ++i)  
    if (an[i] == 1) // i 留存，是质数  
        for (int j = i*2; j < NUM; j += i) // i 的倍数  
            an[j] = 0; //划掉
```



外层循环何时结束？

可用 NUM 作为界限： $i \leq \text{NUM}$

仔细分析不难发现，只要 i 超过NUM的平方根，就可以划掉到既定范围内的所有合数： $i * i \leq \text{NUM}$

```
int main () { //筛法求质数
    const int NUM = 200;
    int an[NUM+1];
    int i, j;
    //数组初始化：用1表示未划掉，用0表示已被划掉
    //an[0] = an[1] = 0; //0和1不是质数
    for (i = 2; i <= NUM; ++i) //开始时数组元素设置为1
        an[i] = 1;
    //筛法
    for (i = 2; i * i <= NUM; ++i) //不用 sqrt 函数
        if (an[i] == 1)
            for (j = i*2; j <= NUM; j += i)
                an[j] = 0;
    //输出
    for (i = 2, j = 0; i <= NUM; ++i)
        if (an[i] != 0)
            cout << i << ((++j) % 10 != 0 ? '\t' : '\n');
    cout << endl;
    return 0;
}
```

小结：

由上面两个程序可知，在程序中使用数组时，一定要想好
元素的下标 和 **元素的值** 各代表什么含义。

例6-3中，元素的下标表示相应的 数字字符；
元素的值表示数字字符的出现次数。

例6-4中，元素的下标表示相应的数字；
元素的值表示它是否留存（1为留存，0为划掉）。

当然，具体含义可以根据自己的想法来进行设置。

6.2.3 约瑟夫问题

【例6-5】“约瑟夫环”是一个数学上和计算机编程上的经典问题：设有 n 个人（以编号 $1, 2, 3, \dots, n$ 分别表示）围成一个圆圈，从编号 m 的人开始由 1 开始报数，每个正好报到数 k 的人退出游戏，后面的一个人重新由 1 开始报数，数到 k 的那个人又出列；依此规律重复下去，直到只剩下最后一人。求最后剩下那个人的编号。

首先要考虑数据的表示。很显然需要使用一个数组。

如何表示人员的编号和人员的去留？

——只能使用数组元素的下标和数组元素的值来分别表示。

可以有多种方法。按照不同的表示方法，可以设计出不同的编程算法。

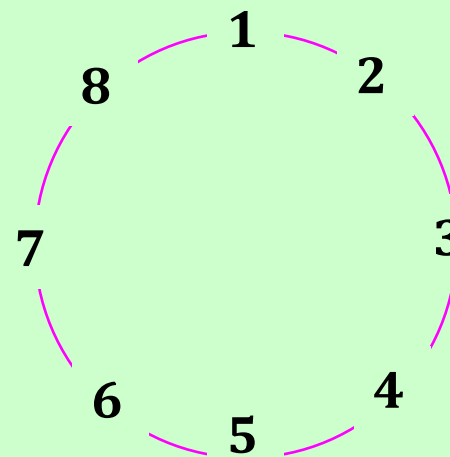
下面选择用数组元素下标表示所有人员的编号、用元素的值表示人员的去留。

题目要求把 n 个人编号为 $1 \sim n$ 来表示。

(假如只定义一个长度为 n 的数组，则需要用下标 $0 \sim n-1$ 来表示 $1 \sim n$ 这些数字，程序中容易出错。)

定义一个长度为 $n + 1$ 的数组，并把下标为 0 的元素闲置不用。使元素下标与所表示的值直接对应。

线性排列可以视为环形排列：



可以用 1 表示留存、用 0 表示退出。

有必要把**所有人的出列顺序**都记录下来。

因此，决定用 0 表示留存，用大于 0 的正数表示依次退出的序号（程序中需要用**一个变量来记录**）。

循环过程：

需要一个变量记录报数的数值。

在按照圆圈循环报数时，需要检查数组的元素值：
值为 0 则报数加1，值大于 0 则报数不变。

最终结果：

求“最后剩下那个人的编号”，可以调整为求出
“最后退出的那个人”的编号即可。

方便观察：

每退出一人就全部输出数组所有元素的值。

```

int main() {
    enum { n = 8 }; //总人数
    int h[n + 1] = {0}; //定义数组并赋初值
    //（以元素下标表示人员编号，0号闲置）
    //初值为 0 表示留在圆圈内
    int m = 4, k = 3; //直接给出m和k的初值(从m开始，轮数k)
    int cnt = 0;    //报数时进行计数 counter
    int num = 0;    //退出序号
    int i;
    cout << "Josephus problem solution" << endl;
    cout << "n= " << n << " m= " << m << " k= " << k << endl;
    while (num < n) { //所有人依次退出。最末退出为胜利者
        cnt = (h[m] == 0 ? 1 : 0); //根据h[m] 的值报数 1 或 0
        while (cnt < k) { //进行报数，报数到k
            m++; //m 往后移
            if (m == n + 1) m = 1; //到数组末尾时，重新从编号为1的元素开始
            if (h[m] == 0) cnt++; //此人仍留在圈内，报数增1
            //如果h[m] != 0，则此人已退出，报数不增加。
        }
        num++;
    }
}

```

```
num++; //上面循环结束时，已报数到k，可退出一人
h[m] = num; //此人标记为非0值。写成退出编号以方便观察
for (i = 1; i <= n; i++) //输出数组（不含0号元素），以便观察
    cout << h[i] << (i != n ? '\t': '\n');
}
cout << "last one: " << m << endl; //最末退出者（第n个）即为胜利者
return 0;
}
```

按照题目中给出的 n、m 和 k 值 (n=8, m=4, k=3) , 运行此程序时, 屏幕输出结果为:

Josephus problem solution

n= 8 m= 4 k= 3

0	0	0	0	0	1	0	0
2	0	0	0	0	1	0	0
2	0	0	3	0	1	0	0
2	0	0	3	0	1	0	4
2	0	0	3	5	1	0	4
2	0	6	3	5	1	0	4
2	0	6	3	5	1	7	4
2	8	6	3	5	1	7	4

last one: 2

程序中每退出一人时就把整个数组的值全部输出一次, 由此可以清晰地看到每个人的退出顺序。

例6-4：成绩分类

- 写程序输入一批学生成绩 (≤ 200)。先输出不及格成绩，再输出及格成绩，最后输出不及格和及格的人数。
- 无法在输入过程中完成(除非输入两遍)。用数组记录学生成绩，输入记入数组，而后输出，可避免重复输入。
- 数据装入数组后有许多可能处理方法。
- 最简单的是两次扫描数组，第一次输出不及格成绩，第二次输出及格的成绩。
- 统计工作可在某次扫描中完成（也可在输入中完成），两部分人数之和就是总人数，统计一部分就够了。

```
enum { NUM = 200 };
const double PASS = 60.0;
double scores[NUM];
int main () {
    int i, n = 0, fail;
    while(n<NUM && scanf("%lf",&scores[n])==1)
        ++n; /*输入时需要判断不越界*/
    for (fail = 0, i = 0; i < n; ++i)
        if (scores[i] < PASS) {
            printf("%f\n", scores[i]);
            ++fail;
        }
    for (i = 0; i < n; ++i)
        if (scores[i] >= PASS)
            printf("%f\n", scores[i]);
    printf("Fail:%d\nPass:%d\n",fail, n-fail);
    return 0;
}
```

6.2.4 多项式求值

【例6-6】设数组 p 中存放多项式， $p[i]$ 存放系数 a_i ：

$$a_0 + a_1x + a_2x^2 + \dots + a_kx^k$$

a_0	a_1	a_2	a_3		a_k
-------	-------	-------	-------	--	-------

写程序求 p 表示的多项式在指定点的值。

方法1：求出各项值累加。假设指定点值存于 x ：

```
for (sum = 0.0, n = 0; n < TERMS; ++n) {  
    for (t = p[n], i = 1; i <= n; ++i) t *= x; //计算  $p[n] * x^n$   
    sum += t;  
}
```

循环体内可以改写为（先算 x 的幂）：

```
for (t = 1.0, i = 1; i <= n; ++i) t *= x; //只计算  $x^n$   
sum += t * p[n]; //累加时才乘以乘数
```

可发现有大量重复计算。有必要改进。

通过保存前次的 t 值 (x 的幂次) , 减少重复计算:

```
for(sum=0.0, t=1.0, n=0; n<TERMS; ++n){  
    sum += t * p[n];  
    t *= x;  
}
```

方法2: 多项式可以变形为Horner范型:

$$(((\cdots((a_k x + a_{k-1}) \cdot x + a_{k-2}) \cdots + a_2) \cdot x + a_1) \cdot x + a_0$$

按照这个公式, 求值循环可写为:

```
for (sum = 0.0, n = TERMS-1; n >= 0; --n)  
    sum = sum * x + p[n];
```


主函数采用方法2，采用其它方式的程序请读者作为练习自行写出。

```
int main() {  
    double po[] = {2.1, 3., 5.6, 8.2, 6.4}; //数组储存系数并以示例数据初始化  
    int n = sizeof(po) / sizeof(po[0]); //数组长度  
    double sum, x;  
    int i;  
    cout << "Calculate Polynomial value" << endl;  
    while (1) { //无限循环，内部用 break 退出  
        cout << "Please enter next value for x: ";  
        if ( !(cin >> x)) //用户输入x值（输入非数值时为输入错误）  
            break;  
        for (sum = 0.0, i = n - 1; i >= 0; --i)  
            sum = sum * x + po[i]; //按Horner形式对sum累加  
        cout << "Polynomial value: " << sum << endl << endl;  
    }  
    return 0;  
}
```

6.2.5 定义数组时的考虑

- 如需要一批同类数据，处理中不断修改，就必须使用数组。如要多次检查一批外来数据，一次读取，存入数组在内存中处理，方便/高效。
- 如果数组表示的是全局数据集合，需要在多个函数里使用，那么可考虑定义为外部数组。
- 大的数组应定义为外部，以免占大量运行栈空间。一般系统不允许在函数内定义特别大的数组。
- 若数组保存着递归定义函数的局部数据，就必须定义为自动数组。因为递归调用时需要数组的多份拷贝。
- 其他情况可以根据需要自由选择。

第6章 数组

6.1 数组的定义和使用

6.2 数组程序实例

6.3 以数组为参数的函数

6.3.1 仅调用数组元素值的函数

6.2.2 修改实参数组的元素

6.4 二维和 multidimensional 数组

6.5 字符数组与字符串

6.6 编程实例

6.3 以数组为参数的函数

各种数据对象，都需要考虑与函数的关系。

用函数处理数组，有两种已知的方法：

①元素是基本类型时，将**数组元素**作为函数实参，与将基本类型的变量作为实参一样。

例

局限性：只能处理个别元素，不能批量修改数组元素的值。

② 函数可直接访问**外部数组**。

例

局限性：只能处理确定的外部数组，而且无法让同一函数在不同时刻处理不同的数组。

```
#include <iostream>
using namespace std;
```

```
void prtout(int k) {
    k++;
    cout << k << endl;
}
```

```
int main () {
    int i=0, n[5] = {0, 2, 4, 6, 8};
    for (i=0; i<5; ++i) cout << n[i] << endl; //事先打印一次
    for (int i=0; i<5; ++i)
        prtout(n[i]); //以形参方式访问局部数组的元素
    for (i=0; i<5; ++i) cout << n[i] << endl; //事后打印一次
    return 0;
}
```

方法①：将数组元素
作为函数实参。

局限性：只能处理个别元素；不能批量修改多个数组元素值。

```
#include <iostream>
using namespace std;
```

```
int gn[5] = {0, 2, 4, 6, 8};    // 外部数组
```

```
void prtout2() {
    for (int i=0; i<5; ++i)
        cout << ++gn[i];
}
```

方法②：函数直接访问
外部数组

局限性：只能处理特定
的外部数组。

```
int main () {
    int i=0;
    for (i=0; i<5; ++i) cout << gn[i] <<endl; //事先打印一次
    prtout2(); //调用 prtout2函数。注意需要写括号。
    for (i=0; i<5; ++i) cout << gn[i] <<endl; //事后打印一次
    return 0;
}
```

假设有多个数组都需要求平均值。上面方法都不合适。

需要定义以数组为参数的函数。

以数组为参数，处理整个数组不再是问题。

不同调用可以完成对不同数组的计算。

定义以double数组为参数，求元素平均值的函数，就可解决所有double数组的平均值问题

6.3.1 仅调用数组元素值的函数

C和C++允许函数有数组形参，描述方式是在参数名后面写一对方括号，其中不必写数组的大小。

实际编写函数时，还需要把数组的大小也传递给函数。通常是用另一个参数传递数组的大小。

【例6-7】编写一个求元素平均值的函数，该函数需要从参数中获得数组名称和数组长度。

```
double avrg(int len, double a[]) {  
    double sum = 0.0;  
    for (int i = 0; i < len; ++i)  
        sum += a[i];  
    return sum / len;  
}
```


调用上述函数：

```
int main () {  
    double b1[3] = {1.2, 2.43, 1.074};  
    //定义数组并以示例数据初始化  
    double b2[5] = {6.54, 9.23, 8.463, 4.25, 0.386};  
    cout << "averages: " << avrg(3, b1) << "\t" << avrg(5, b2)  
    << endl;  
    return 0;  
}
```

使用时必须关注越界问题。

例：avrg(5, b1)引起数组越界（b1只有3个元素）。

长度实参可小于数组大小，avrg(3, b2) 求 b2 前 3 个元素的平均值。可将数组前段当作数组使用。

6.3.2 修改实参数组的元素

通常情况：函数参数都是值参数，形参是函数体的局部变量。调用时将实参的**值**送给形参。

以数组为参数的函数**不是**普通的值参数机制。



以数组为参数的函数在执行时，对**形参**数组元素的操作，就是**直接**对函数调用时的**实参**数组元素的操作。

这样定义函数，就可以改变实参数组元素的值。

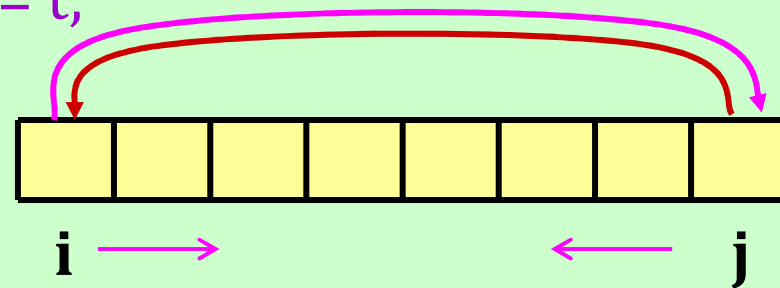
这是数组参数的特殊情况，详情在第7章解释。

【例6-8】 定义函数，翻转参数数组里元素的位置：

```
void reverse(int n, int a[]) {  
    int t, i, j;  
    for (i = 0, j = n-1; i < j; ++i, --j) {  
        t = a[i];  a[i] = a[j];  a[j] = t;  
    }  
}
```

写程序检查 reverse 的效果。

```
int main () {  
    int i, b[] = {1, 2, 3, 4, 5, 6, 7};  
    int n = sizeof(b)/sizeof(b[0]);  
    for (i = 0; i < n; i++) cout << b[i] << (i==n-1 ? "\n" : "\t");  
    reverse(n, b); //!!!  
    cout<< endl <<"After reverseersion:" << endl;  
    for (i = 0; i < n; i++) cout << b[i] << (i == n-1 ? "\n" : "\t");  
    return 0;  
}
```



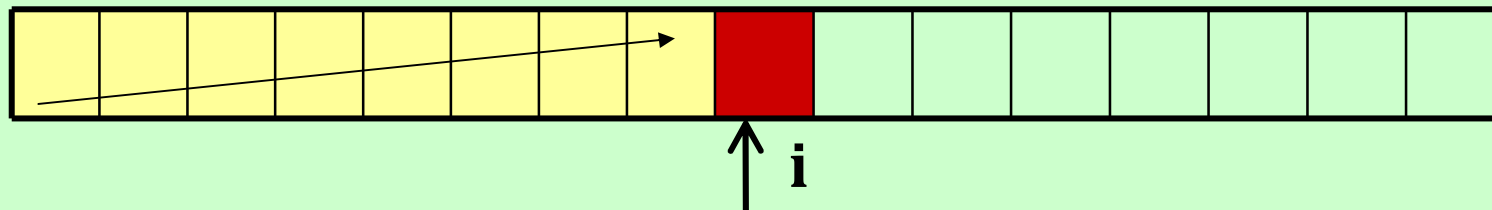
【例6-9】数组元素排序(修改实参数组的元素)。

对一组数据经常要做的一个操作是把它们按照某种标准重新排列，这种操作被称为排序 (sorting) 。

假定有 n 个数值存放在数组 a 中，写一个函数，把它们从小到大重新排列。

排序的方法很多，这里介绍直接插入排序。

假设左边那部分数据已经排好序（最开始时，第一个元素已排好序），右边是尚未排序的元素。



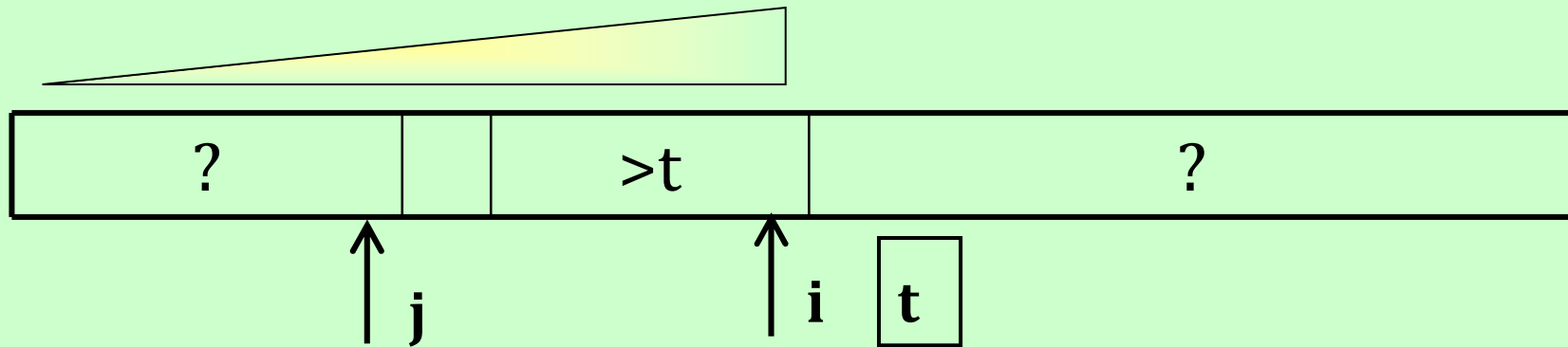
下标 0 到 $i-1$ 的部分已排序。反复把 i 处的数据插入已排序部分。随着 i 增大右段越来越短，最终所有数据都排好。

考虑元素 $a[i]$ 如何插入到 $a[i-1]$ 到 $a[0]$ 之间：

先把 $a[i]$ 存入临时变量 t 里，于是 $a[i]$ 闲置。

用 t 逐个与 $a[i-1]$ 到 $a[0]$ 比较：变量 j 从 $i-1$ 递减到 0。

- 如果元素 $a[j]$ 比 t 要大，就后移一位（于是 $a[j]$ 闲置）。
- 当 $a[j]$ 不大于 t 时，正好把 t 放入空位， a 里的前 i 个值就排好序了。



开始令左段只有一个元素（单元素序列总已排序），排序过程可如下描述：

```
for (i = 1; i < n; ++i) {  
    把a[i]的值插入a[0]到a[i-1]一段里的  
    正确位置， 保持其他元素顺序不变  
}
```

```
void insertsort(int n, int a[]) { //整型数组按照递增序直接插入排序  
    int i, j;  int t;  
    for (i = 1; i < n; ++i ) {  
        for (t = a[i], j = i - 1; j >= 0 && t < a[j]; --j)  
            a[j+1] = a[j];  //大元素依次后移  
        if (j != i-1)  
            a[j+1]=t;  
    }  
}
```

用于测试此排序函数的辅助打印函数和主函数如下：

```
void printall(int n, int a[]) {  
    for (int i = 0; i < n; ++i)  
        cout << a[i] << '\t';  
    cout << endl;  
}
```

```
int main() {  
    int arr[] = {3, 2, 5, 1, 7, 8, 10, 6, 5 }; //定义数组并初始化  
    int n = sizeof(arr) / sizeof(arr[0]);  
    cout << "before sorting:" << endl;  
    printall(n, arr);  
    insertsort(n, arr);  //!!!  
    cout << "after sorting:" << endl;  
    printall(n, arr);  
    return 0;  
}
```

应该在 IDE 中对此程序进行调试，
在调试过程中观察变量的值，
以便深刻地理解直接插入排序算法。

第6章 数组

6.1 数组的概念、定义和使用

6.2 数组程序实例

6.3 数组作为函数参数

6.4 二维和 multidimensional 数组

6.4.1 多维数组的初始化

6.4.2 多维数组的使用

6.4.3 多维数组作为函数的参数

6.4 字符数组与字符串

6.5 编程实例

6.4 两维和多维数组

一维数组有一个下标，元素线性排列。

需要处理更复杂的结构，如线性代数中的二维矩阵。

可以定义两维/多维数组：

```
int a[3][2]; // 3×2的 int 数组  
double b[4][4]; // 4×4 的 double 数组  
int a1[3][2][4]; // 三维数组
```

两维数组看作一维数组的数组，元素的成员类型和成员个数都相同。

例如 a 是长度为3、成员为长度为2的一维数组。

6.4.1 多维数组的初始化

定义时直接初始化:

```
int a[3][2] = {{1, 2}, {3, 4}, {5, 6}};
```

内嵌括号初始化成员数组。不足时自动置0。

也可不写内嵌括号:

```
int a[3][2] = {1, 2, 3, 4, 5, 6};
```

初始值按顺序赋给基本成分，不够时置0。

若初始化给出了全部元素值，第一下标的元素个数可以不写（可根据初始化表示算出）。

```
int a[][2] = {{1,2},{3,4},{5,6}};
```

```
int b[][2] = {1, 2, 3, 4, 5, 6};
```

6.4.2 多维数组的表示和使用

设有数组定义：

```
int a[3][2];  
double b[4][4];
```

a 表示整个数组，a[0]、a[1] 和a[2]表示成员数组。

a[0][1] 表示 a 的 0 成员数组中下标 1 的元素。

例：

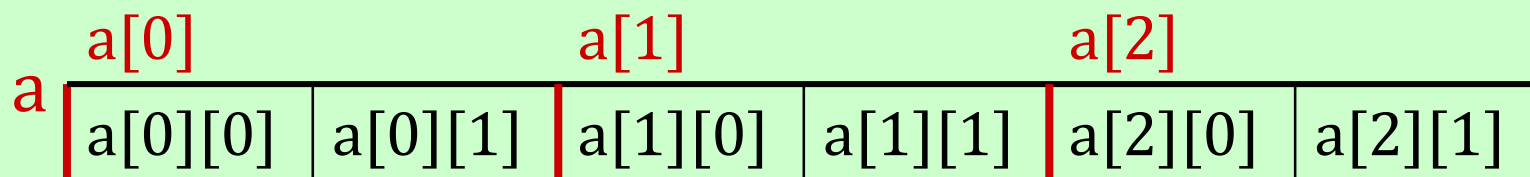
```
a[2][1] = a[0][1] + a[1][1];
```

```
for (i = 0; i < 4; ++i)  
    for (j = 0; j < 4; ++j)  
        b[i][j] = i + j;
```

(语言细节)

数组元素连续存储，多维数组一样，依次存储成员数组，成员数组按同样方式表示。

二维数组 $a[3][2]$ 的内部表示：



注意：a 的开始位置也是其首成员 $a[0]$ 的开始位置，也是 $a[0]$ 首成员 $a[0][0]$ 的位置。

C/C++ 的数组存放方式，一行（成员数组）元素连续存储，这种形式又称按行方式或行优先方式。

【例6-10】写程序段求出由两维数组A、B表示的 4×4 矩阵的乘积，存入两维数组C（设A，B已有值）并输出。

```
for(i = 0; i < N; ++i)
    for(j = 0; j < N; ++j)
        for(C[i][j]= 0.0, k = 0; k < N; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

```
for(i = 0; i < N; ++i)
    for(j = 0; j < N; ++j)
        cout << C[i][j] << (j == N-1 ? '\n' : '\t');
```

6.4.3 多维数组作为函数参数

多维数组作为函数参数时不必给出最左一维长度，但要求给出除最左一维外其他各维的长度。

【例6-11】 求出 $n \times 5$ 数组中数据均值(n 是参数):

```
double aaverage (int n, double a[][5]){  
    int i, j;  
    double sum = 0.0;  
    for (i = 0; i < n; ++i)  
        for (j = 0; j < 5; ++j)  
            sum += a[i][j];  
    return sum / (5 * n);  
}
```

可用于任何 $n \times 5$ 的数组，但不能用于例如 4×7 的数组
下章将介绍一种定义通用函数的方法（利用指针）。

第6章 数组

6.1 数组的概念、定义和使用

6.2 数组程序实例

6.3 数组作为函数参数

6.4 二维和 multidimensional 数组

6.5 字符数组与字符串

6.5.1 字符数组

6.5.2 字符串

6.5.3 程序实例

6.5.4 标准库字符串处理函数

6.5.5 输出文本里的最长行

6.6 编程实例

6.5 字符数组与字符串

6.5.1 字符数组

C/C++ 常用于写文字处理程序，文字信息以字符形式存在字符数组里。对字符数组有一些特殊机制。

字符数组定义方式（与其他数组一样）：

```
char line[1000];
```

字符数组可在定义时初始化：

```
char city[15]={'B','e','i','j','i','n','g'};
```

未指定初值的元素自动设为编码为 0 的字符。

→ 0字符/空字符，用'\0'表示，有特殊用途。

6.5.2 字符串

字符串 写法: "字符串文字量"。

字符串不能跨行。如果多个字符串之间仅由空白分隔, 系统会将它们连成一个长字符串。

不能直接写出的字符采用换意序列: \n \t \"

字符串是由系统自动地以字符数组形式保存, 存储形式是在所有字符后放 '\0' 作为串结束标志。



例: `cout << "Beijing";`

B	e	i	j	i	n	g	\0
---	---	---	---	---	---	---	----

'\0' 不是串内容, 却是字符串表示不可缺的部分。

标准库的字符串处理函数都按这种表示设计, 写字符串处理程序时也应该遵守这一规则。

问：能在定义的字符数组变量里存放字符串吗？

字符数组里至少存放一个空字符，就符合字符串形式，可当作字符串使用。→ 数组里存放着字符串。

有多个空字符时，以从左向右第一个认为是字符串结束符。

```
char a[5] = {'i','s','n','o','t'}, //不是字符串
```

```
    b[5] = {'g','o','o','d','\0'}, //是字符串
```

```
    c[5] = {'f','i','n','e'}, //是字符串
```

```
    d[5] = {'o','k','\0'}, //是字符串
```

```
    e[5] = {'o','k','\0','x'}; //是
```

可用字符串指定字符数组的初值（是简写）：

```
    char a1[20] = "Peking University";
```

未标元素个数时，数组大小确定为串长加1。

```
    char a3[] = "Peking University"; //18个字符元素
```

6.5.3 字符串的输出与输入

字符串（以双引号括起来的字符串常量，或者存放了字符串的字符数组）的输出可用两种方法：

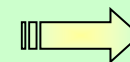
逐字符输出：`cout <<`, `putchar`, `cout.put`

整个字符串一次性输出：`cout <<`

【例6-12】测试用多种方法输出字符数组或字符串。每次输出之后故意添加输出字符'#'，以便观察各种输出方法的差异。

```
int main () {  
    const int LEN=20;  
    char str[LEN] = "Hello, world!"; //定义字符数组str并初始化。  
    //注意，这样初始化的字符数组在字符后面部分自动填充为多个空字符  
    for (int i = 0; i < LEN; i++) //输出方法1：逐字符输出  
        putchar (str[i]); //或cout.put(str[i]); 或 cout << str[i];  
    cout << '#' << endl;
```

Hello, world! #



```
cout << str << '#' << endl; //输出方法2：字符串整体输出
```

```
Hello, world!#
```

```
str[4] = '\0'; //把第5个字符'o'改为空字符
```

```
for (int i = 0; i < LEN; i++) //输出方法1：逐字符输出
```

```
    putchar (str[i]); //或cout.put(str[i]); 或 cout << str[i];
```

```
cout << '#' << endl;
```

```
Hell , world!      #  
Hell#
```

```
cout << str << '#' << endl; //输出方法2：字符串整体输出
```

```
return 0;
```

```
}
```

小结：

逐字符输出时，空字符被输出为空格形式。

整体输出时，遇到第一个空字符就结束输出。

输入字符串时，也有多种方法可以逐字符输入或整体输入。但是需要注意的是，有些方法不能接收空格，有些方法则可以接收空格。如果在程序中所需要输入的字符串中含有空格（例如英文姓名全称或者英文书名），那就必须选用可以接收空格的输入方法。

逐字符输入时，可以用三种方法：

- (1) getchar()函数；
- (2) cin.get()函数；
- (3) “cin >>”流式输入(不能接收空格)。

最容易想到的逐字符输入一个字符串的方法：

```
for (int i = 0; i < LEN-1; i++) // i 最大值为 LEN - 2  
    str[i] = getchar(); //str[i] = cin.get(); //cin >> str[i];  
str[i] = '\0'; //添加空字符作为字符串结束符
```

缺点是必须输入 LEN-1 个字符。

改进，读取字符同时判断是否为指定结束字符：

```
int i;  
for (i=0; i < LEN-1 && (str[i] = getchar())!='\n'; i++)  
    //cin.get()亦可  
    ; //空循环体  
str[i] = '\0'; //添加空字符作为字符串结束符
```

这个循环将一行字符读入到数组 str，以接受到回车键作为输入结束（用户可以改为 EOF 甚至其它字符）。

注意，用条件 $i < \text{LEN} - 1$ 保证对数组 str 的访问不越界。读入完毕后，i 记录着读入字符的个数，而且我们需要在字符串末尾添加空字符作为字符串结束符。

对字符串**整体输入**时，简单的方法是这样：

```
cin >> str;
```

缺点：不能接收空格，遇到空格就会认为是字符串结束。

要想让字符串整体输入时接受空格字符，就需要使用C++ 提供的 cin 流中的 **getline()** 函数：

```
cin.getline(字符数组名str, 字符个数n, 结束符'\n')
```

功能：一次连续读入多个字符（可以包括空格）直到读满n个或遇到指定的结束符（默认为回车符）为止。读入的字符串存放于字符数组str中，但不存储结束符。自动在字符串末尾添加空字符。

例：

```
cin.getline(str, LEN);
```


还需要注意：由于系统是**缓冲式输入**，因此上一次输入时最末键入的多余字符或回车符可能被意外地被下一次输入读到。

因此在**连续多次输入**时，需要注意把上一次输入的字符清除掉。

需要用到 `cin.clear()` 和 `cin.sync()` 这两个函数，清除输入缓冲区中残留的输入数据。

当然，也可以使用字符串类或文件作为输入源（参见“4.3.4 字符串流与文件流输入输出”），这时可以类似地使用相应的 `get()` 函数和 `getline()` 函数。

【例6-13】分别用 `getchar()` 和 `cin.getline()` 方法输入字符串，并注意在前后两次输入时清除输入缓冲区。

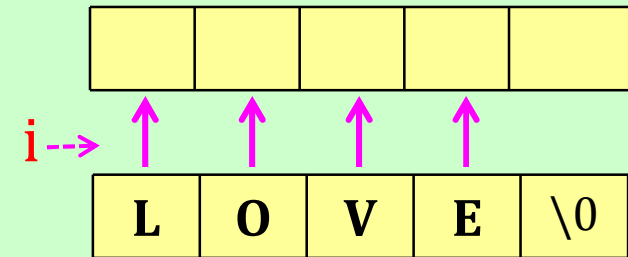
```
int main () {  
    const int LEN = 10;  
    char str[LEN];  
    cout << "Input string with getchar: ";  
    int i;  
    for (i=0; i<LEN-1 && (str[i] = getchar())!='\n'; i++)  
        ; //空循环体  
    str[i]='\0';  
    cout << "str: " << str << endl << endl;  
    cin.clear(); //清除状态标记  
    cin.sync(); //清空缓冲区  
    cout << "Input string with cin.getline: ";  
    cin.getline(str, 20); //输入时默认以回车符为结束符  
    cout << "str: " << str << endl;  
    return 0;  
}
```

当第一次输入的字符数超过字符数组 `str` 的长度 `LEN` 时，多余的字符可能会自动地成为第二次的输入。只有使用这两个函数，第二次输入才不会受到前一次输入的影响。

6.5.4 字符串处理程序实例

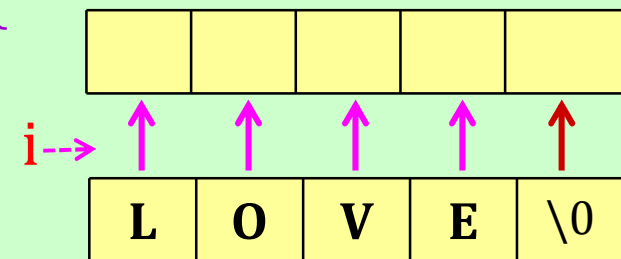
【例6-14】 写函数将一字符串复制到一字符数组。假定数组足以存放被复制串及空字符。定义：

```
void str_copy (char s[], const char t[]) { //string, template
    int i = 0;
    while (t[i] != '\0') { s[i] = t[i]; ++i; }
    s[i] = '\0';
}
```



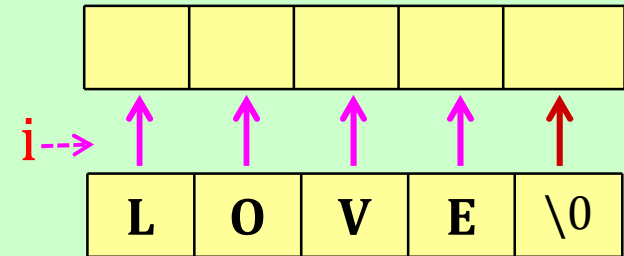
循环结束时t[i]的空字符正好赋给s[i]。赋值有值：

```
void str_copy (char s[], const char t[]) {
    int i = 0;
    while ((s[i] = t[i]) != '\0') ++i;
}
```



进一步简化:

```
void str_copy (char s[], const char t[]) {  
    int i = 0;  
    while (s[i] = t[i]) ++i;    //最后一个值就是 \0  
}
```



➔ 教师演示编写用于调用的主程序，并作调试

字符串处理函数的典型循环:

```
for (i = 0; s[i] != '\0'; ++i) {.....}
```

```
for (i = 0; s[i]; ++i) {.....}
```

用是否空字符确定对字符串的处理是否已完成。

【例6-15】（二进制转换）写函数，给它一个表示二进制数的0/1字符串，它算出这个串表示的整数值。

二进制数 $b_n b_{n-1} \dots b_2 b_1 b_0$ 的值可以用下面公式计算：

$$(((\dots((b_n \times 2) + b_{n-1}) \times 2 + \dots) \times 2 + b_2) \times 2 + b_1) \times 2 + b_0$$

据此可写出循环，循环条件判断二进制数是否结束。

二进制数位只能是0/1，只需要在遇到1时加一。

```
int bin2int(const char s[]) {  
    int i = 0, n = 0;  
    for (i=0; s[i]!='\0'&&(s[i]=='0' || s[i]=='1'); ++i){  
        n = n * 2;  
        if (s[i] == '1') ++n;  
    }  
    return n;  
}
```

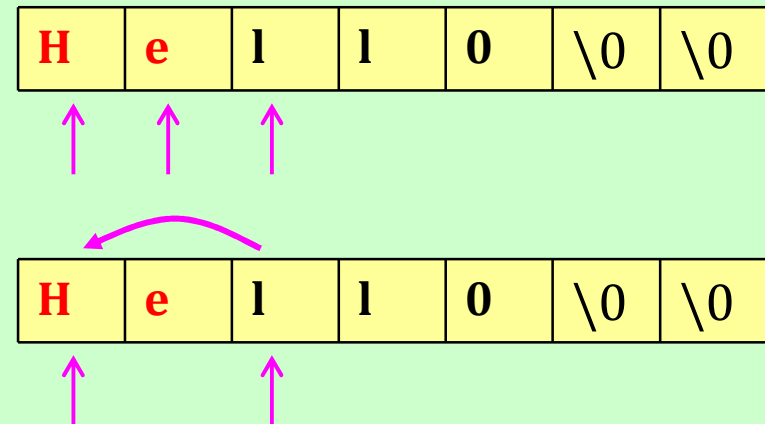
由于数字字符的编码连续排列，人们常用：

```
int bin2int(const char s[]) {  
    int i, n = 0;  
    for(i=0; s[i]!='\0'&&(s[i]=='0' || s[i]=='1') ++i)  
        n = n * 2 + (s[i] - '0');  
    return n;  
} // 这种写法可以推广到十进制和其他进制
```

【例6-16】考虑一个在保存字符串的数组里操作的例子。假定需要一个函数，它处理保存着字符串的数组，它删除字符串的前 n 个字符，留下删除后的字符串。

先找到要保留的第一个字符位置，然后把从那里开始的有效字符逐个拷贝到前面来：

```
void delPrefix(int dnum, char a[]) {  
    int i = 0, j = 0;  
    while (i < dnum && a[i] != '\0')  
        ++i;  
    while (a[j] = a[i]) {  
        ++i; ++j;  
    }  
}
```



6.5.4 标准库字符串处理函数

<cstring> 里描述各种字符串处理函数，常用：

1. 字符串长度函数 `strlen(const char s[])`
2. 复制函数： `strcpy(char s[],const char t[]);`

t 应为字符串，const说明参数值不修改。

字符数组 s 应足够大，以保证复制不越界。例：

```
char a1[20], a2[20];  
strcpy(a1, "programming");  
strcpy(a2, a1);
```

限界复制函数 `strncpy`，限制复制长度：

```
strncpy(char s[],const char t[],int n);
```


3. 字符串连接函数

`strcat(char s[], const char t[]);`

把字符串 t 复制到字符数组 s 已有的字符之后，形成连接起来的串。数组 s 必须足够大。

```
char b1[40] = "Programming", b2[10];  
strcat(b1, " language");  
strcpy(b2, " C"); strcat(b1, b2);
```

另有限界连接函数 `strncat`:

`strncat(char s[], const char t[], int n);`

4. 字符串比较函数

`int strcmp(const char s1[], const char s2[])`

- 两字符串相同时返回值 0;
- s1 大于 s2 的情况下返回大于 0 的值;
- 否则返回小于 0 的值。
- 判断字符串大小的标准是字符串的字典序（普通英语词典里单词排列的顺序）

限界比较函数：

`int strncmp(const char s1[], const char s2[], int n);`

5、在字符串中查找字符

`char * strchr(char s[], int ch)`

查找字符串 s 中首次出现字符 ch 的位置，返回首次出现 ch 的位置的指针，返回的地址是字符串在内存中随机分配的地址再加上所搜索的字符在字符串位置，如果 s 中不存在 ch 则返回 NULL。

6、在字符串中查找字符串

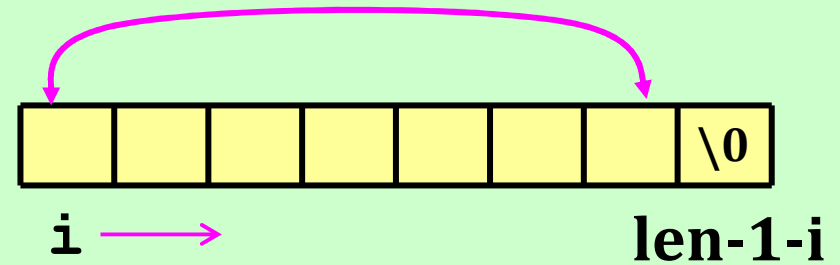
`char * strstr(const char str1[], const char str2[])`

查找目标字符串 待查找字符串

该函数返回 str2 第一次在 str1 中的位置，如果没有找到，返回 NULL。

【习题6-13】回文(palindrome)是从前向后和从后向前读都一样的词语或句子。例如英文的“madamimadam”，其原文是“Madam, I'm Adam”。写一个函数，判断一个英文字符串是否为回文。

与教材中的 reverse 函数类似

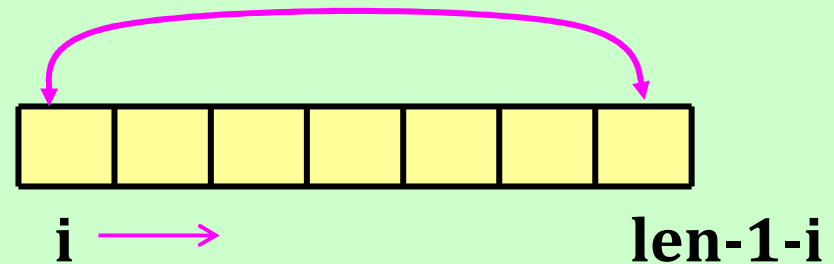


```
int palindrome(char str[]) {  
    //函数功能：判断全英文的字符数组 str[] 是否回文  
    //说明：形参为一个字符数组（不需要长度参数）  
    int len = strlen(str); //使用标准库函数 strlen 获得字符串长度  
    //逐个比较字符串的头部元素和相应的尾部元素  
    for (int i = 0; i < len/2; i++)  
        if (str[i] != str[len-1-i])  
            return 0; //只要发现一对字符不相等，则可判断不是回文  
    return 1; //循环结束，所有相应元素都相等，从而判断是回文  
}
```

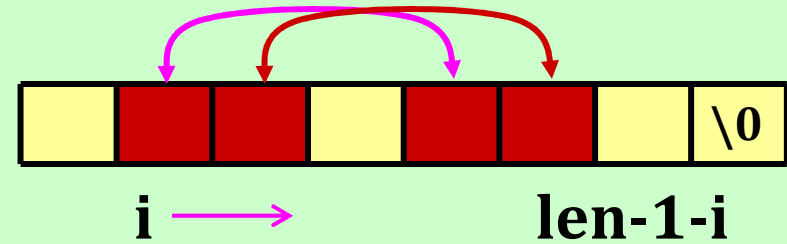
写程序检查效果

```
int main() {  
    //定义字符数组 s 并初始化为纯英文示例字符串  
    char s[100] = "madamimadam";  
    //或用 strcpy 把常量字符串复制到字符数组  
    //strcpy(s, "wasitacaroracatisaw");  
  
    if (palindrome(s)) //调用函数对 s 进行判断  
        cout << s << " 是回文" << endl;  
    else  
        cout << s << " 不是回文" << endl;  
    return 0;  
}
```

中文也有回文，例如“上海自来水来自海上”。
能否判断包含中文字符的字符串是否回文？



```
int palindrome2(char str[]) {  
    // 函数功能：判断中英文混和的字符数组 str[] 是否回文  
    // 说明：形参为一个字符数组（不需要长度参数）  
    int len = strlen(str); // 使用标准库函数 strlen 获得字符数组长度  
    // 逐个比较字符数组的头部元素和相应的尾部元素  
    for (int i = 0; i < len/2; i++) {  
        if (str[i] >= 0 && str[i] <= 127) {  
            // 英文字符只占一个字节，大小介于 0-127  
            if (str[i] != str[len-1-i])  
                return 0; // 发现一对英文字符不相等，判定不是回文
```



```
}else { //每个中文字符占两个字节，  
//首字节的大小不在 0-127 之间  
    if (str[i] != str[len-1-i-1] && str[i+1] != str[len-1-i] )  
        return 0;  
    i++; // 对于 for 的头部指定的 i，此处比较了两个字符，所以  
    让 i 额外增1  
}  
}  
return 1; //循环结束，所有相应元素都相等，从而判断是回文  
}
```

6.5.6 从文件读取字符串程序实例

【例6-17】 找出(输入文件流)文本中最长行并输出。

基本处理框架：

```
while (还有新输入行)
    if (新行比已记录的最长行更长)
        记录新行及其长度;
输出记录下来的最长行;
```

程序中要用到的数据：

保存已读最长行以输出，字符数组 maxline。

数组 line 保存新行，更长时转存 maxline。

设行长不超过1023字符，用符号常量定义数组大小。

用函数 `cin.getline` 读入一行。

用函数 `strlen` 求得字符串长度

while (还有新输入行)

if (新行比已记录的最长行更长)

记录新行及其长度;

输出记录下来的最长行;

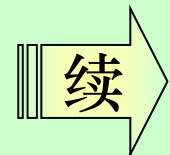
输出字符串

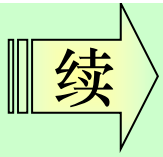
用函数 `strcpy` 复制字符串

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

int main () {
    const int MAXLEN = 1024;
    char line[MAXLEN], maxline[MAXLEN];
    int n, max = 0; // 记录当前行和最长行的长度

    char filename[56] = "plain.txt"; //定义字符数组以存储文件名
    ifstream inFile (filename);    //定义输入文件流并绑定到文件
    if (!inFile) { //如果打开文件失败，则 infile 得到一个零值
        cout << "错误： 未找到数据文件 " << filename << " 。\n";
        exit(1); // 打开文件失败，则显示错误信息并退出程序。
    }
}
```





```
cout << "Reading from file: " << filename << endl << endl;
while (infile.getline(line, MAXLEN))
    if (strlen(line) > max) {
        strcpy(maxline, line);
        max = strlen(line);
    }
infile.close();
cout << "Longest line: \n" << maxline << endl;
cout << "\nLength: " << strlen(maxline) << endl;
}
```

第6章 数组

6.1 数组的概念、定义和使用

6.2 数组程序实例

6.3 数组作为函数参数

6.4 二维和 multidimensional 数组

6.5 字符数组与字符串

6.6 编程实例

6.6.1 拼手气发红包

6.6.2 学生成绩统计分析

6.6.3 统计源程序中的关键字

6.6 编程实例

本节讨论一些基于数组的编程实例。

一个具体问题可写出许多不同的程序。从问题到程序的工作过程中，许多地方需要编程者做出选择。

有些选择涉及对问题的不同考虑或认识，可能引起程序间的显著差异。也有些是具体表达方式的选择。

学习中应努力思考其他解法，应特别注意隐藏在解答后面的选择：考虑了哪些问题，怎样考虑，选择的合理性/不合理性？还可能怎样考虑和处理等。

注意提出这类问题，能从中大大受益。

6.6.1 拼手气发红包

【例6-19】在当代很多社交软件中可以进行“拼手气发红包”活动。发起人指定红包总金额和红包个数，接收者就会得到一个随机金额的红包。编写一个字符界面的程序模拟这一过程，着重考虑如何把总金额 t 元随机地发放给 n 个人（人民币的最小金额单位是0.01元，设置参数时应满足 $t \geq n * 0.01$ ）

【分析】事先把总金额以随机方式分解为 n 个数，保存在一个数组中，然后在每个人领取时依次取到这 n 个数（直接输出）。因此可以产生 n 个随机数，然后进行归一化，映射到总金额为 t 。

所产生的随机数应该如何分布为好？简单方式：所产生的随机数对称地分布于平均值 t/n 上下，例如 0.5 倍至 1.5 倍之间。

归一化时，最后一个红包是总金额减去前 $n-1$ 个红包。

在程序中以“分”为单位，按整数处理。

在最高层，可以将这一程序的工作分为三个步骤：设置红包参数，用随机数进行分配，输出红包金额。

做出程序的第一级分解，写出函数原型说明和主函数：

```
#include <iostream>
```

```
#include <cstdlib>    //使用随机数函数所需的库文件
```

```
#include <ctime>      //使用 time() 函数所需的库文件
```

```
using namespace std;
```

```
int setpara(int &t, int &n, int MAX);
```

```
//设置红包总金额t和人数 n (<=MAX)
```

```
int allocate(int t, int n, int money[]);
```

```
//总金额t分配到长为n的数组money中
```

```
void printout(int t, int n, int money[]);
```

```
//输出money数组中的红包金额
```

```
int main() {  
    const int MAX = 100;  //最大人数  
    int t, n, money[MAX];  
    cout << "== 拼手气发红包 ==" << endl;  
    if (!setpara(t, n, MAX)) {  //设置参数，若失败则退出  
        cout << "参数设置失败，程序结束。";  
        exit(1);  
    }  
  
    allocate (t, n, money);  //分配红包金额  
    printout (t, n, money);  //打印输出  
    return 0;  
}
```



```

int setpara(int &t, int &n, int MAX){ //设置红包总金额t和人数 n (<=MAX)
    double total; //红包总金额 (元)
    int errs = 0, ERRNUM = 3;
    do {
        cout << "请输入红包总金额(元): ";
        cin >> total;
        cout << "请输入红包个数: ";
        cin >> n;
        if (n > MAX) {
            cout << "错误: 总人数超过了 "<< MAX << endl;
            errs++;
        }
        t = total * 100; //货币单位由元转换为分
        if (t < n) { //如果红包总金额分数小于人数
            cout << "错误: 红包总金额分数小于人数! 请重新输入。" << endl;
            errs++;
        }
    } while ((n > MAX || t < n) && errs <= ERRNUM);
    return (errs <= ERRNUM ? 1 : 0);
}

```

```

int allocate(int t, int n, int money[]){//总金额t分配到长为n的数组money中
    int i, total = 0;
    int min = 0.5 * t / n, max = 1.5 * t / n;
    srand(time(0));
    for (i = 0; i < n; i++) { //产生 n 个随机数
        money[i] = rand() % (max - min + 1) + min;
        total += money[i];
    }
    //对红包金额进行归一化。
    //为了防止数值误差，只对前n-1个进行归一化，最后一个以减法赋值
    int sum = 0;
    for (i = 0; i < n-1; i++) { //前 n-1 个进行归一化
        money[i] = double(money[i]) / total * t;
        sum += money[i];
    }
    money[n-1] = t - sum;
    //TODO: 额外调整避免0值
    return 0;
}

```

```

void printout(int t, int n, int money[]){
    //输出money数组中的红包金额
    for (int i = 0; i < n; i++)
        cout << 0.01 * money[i] << (i % 5 == 4?
        '\n':'\t'); //单位为"元"。
}

```

发红包程序的改进：避免发 0 元红包

6.6.2 学生成绩统计分析

【例6-20】文件里保存着一批学生成绩，写程序读入这些成绩，先分别输出不及格的成绩和及格的成绩，再输出不及格和及格的人数统计，计算其平均值 M 和标准差 S ，并做直方图（文本窗口中横向作图）。

$$M = \frac{1}{N} \sum_{i=1}^N x_i$$

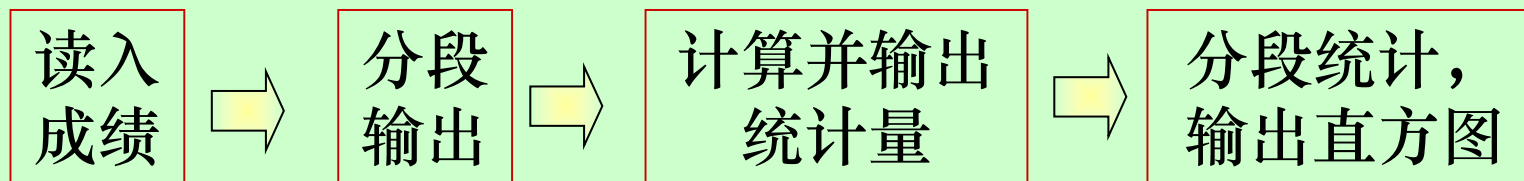
$$S^2 = \frac{1}{N-1} \sum_{i=1}^N (x_i - M)^2$$

程序中需要反复使用这批数据，应存入**数组**。

程序工作比较多，考虑将主要工作**划分为若干函数**。

用一个双精度数组保存所有成绩，作为基本数据。

在最高层可将程序工作分为四步（第一层分解）：



```
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <iomanip>
using namespace std;

int readscores(int lim, double tb[]);
//输入不超过lim个数据到tb，返回项数
void printout (int num, double tb[], double passline);
//分段输出
void statistics(int num, double tb[]);
//统计数组tb里的num个数据项
void histogram(int num, double tb[], int high);
//绘制最高为high的直方图
```

```
int main() {  
    enum { NUM = 200, HISTOHIGH = 60 }; //最大数据项数和  
        直方图的最大高度  
    const double PASSLINE = 60.0; //及格分数线  
    double scores[NUM]; //学生的成绩分数  
  
    int n = readscores(NUM, scores); //读入最多NUM个学生  
        成绩，n为学生人数  
    printout(n, scores, PASSLINE); //分段输出  
    statistics(n, scores); //计算并输出统计量  
    histogram(n, scores, HISTOHIGH); //分段统计并输出直方图  
    return 0;  
}
```

输入函数的一种实现:

```
int readscores(int limit, double tb[]) {  
    int i = 0;  
    //手工输入  
    cout << "input scores:" << endl;  
    while (i < limit && cin >> tb[i])  
        ++i;  
    //从数据文件中读取  
    // char fname[20]="scores.txt";  
    // ifstream infile(fname);  
    // if (!infile) { cout << "ERROR: " << fname << endl; exit(1); }  
    // cout << "read data from input file:" << fname << endl;  
    // while( i < limit && infile >> tb[i])  
    //     i++;  
    //随机数模拟, 分数区间为[30, 100]  
    // srand(time(0));  
    // for (i=0; i< limit ; i++)  
    //     tb[i] = 30 + rand()%(100-30+1);  
    // cout << "Generated random scores : "<< i << endl;  
    return i;  
}
```

```
void printout (int num, double tb[], double passline) { //按及格线分段输出
    int i, fail, pass;

    cout << "Failed socres: " << endl;
    for (fail = 0, i = 0; i < num; ++i)
        if (tb[i] < passline) {
            ++fail;
            cout << tb[i]<< (fail % 10 != 0 ? '\t': '\n');
        }
    cout << endl << "Total failed: " << fail << endl << endl;
    cout << "Passed socres: " << endl;
    for (pass = 0, i = 0; i < num; ++i)
        if (tb[i] >= passline) {
            pass++;
            cout << tb[i]<< (pass % 10 != 0 ? '\t': '\n');
        }
    cout << endl << "Total passed: " << pass << endl;
}
```



```

void statistics(int n, double tb[]) {
    int i;
    double s, sum, avr;
    if (n < 1) {
        cout << "Data too few. Can't perform statistics.\n";
        return;
    }
    for (sum = 0.0, i = 0; i < n; ++i)
        sum += tb[i];
    avr = sum/n;
    for (sum = 0.0, i = 0; i < n; ++i)
        sum += (tb[i] - avr) * (tb[i] - avr);
    s = sqrt(sum / (n - 1));
    cout << "Total students: " << n << endl;
    cout << "Average score: " << fixed << setprecision(2) << avr << endl;
    cout << "Standard deviation: " << s << endl << endl;
}

```

$$M = \frac{1}{N} \sum_{i=1}^N x_i$$

$$S^2 = \frac{1}{N-1} \sum_{i=1}^N (x_i - M)^2$$

直方图生成（用横向的直方图）：

每个成绩段输出一组字符，选H作为基本字符。为描述的方便，定义一个字符序列输出函数：

```
void prtHH(int n) {  
    int i;  
    for(i = 0; i < n; ++i) putchar('H');  
}
```

分段长度可用符号常量表示，根据它可算出分段数。定义两个常量，利用它们之间的关系：

```
enum { SEGLEN = 5, HISTONUM = (100/SEGLEN)+1 };
```

分段成绩数统计：用数组统计各分段成绩人数，数组命名为 segs，其中应有 HISTONUM 个计数器。

处理的是等长分段，存在从成绩得到计数器下标的简便方法。对 scores 的元素正确更新相应计数器：

```
segs[((int)scores[i])/SEGLLEN]++;
```

将成绩强制转到int后除以分段长度得到计数器下标。

下面考虑用如下形式输出直方图行：

```
< 80: 23|HHHHHHHHHHHHHHH
```

为使直方图规范化（最长行 HISTOHIGH 个字符），还要求出最长行的长度。

```

void histogram(int n, double tb[], int high) {
    int i, mx;
    enum {SEGLen = 10, HISTONUM = (100 / SEGLen) + 1 };
    int segs[HISTONUM];

    if (n == 0) return;
    for (i = 0; i < HISTONUM; ++i) segs[i] = 0; // 初始化
    for (i = 0; i < n; ++i) // 统计各分段人数
        segs[(int)tb[i] / SEGLen]++;
    for (mx = 1, i = 0; i < HISTONUM; ++i) // 为规范化找出最大个数
        if (segs[i] > mx)    mx = segs[i];

    cout << "Histogram: " << endl;
    for (i = 0; i < HISTONUM; ++i) { //输出
        cout << setw(3) << (i) * SEGLen << '~ :' << setw(4) << segs[i] << '|';
        prthH(segs[i] * high / mx);
        cout << endl;
    }
    cout << endl;
}

```

分析和改进

若文件中都是0~100的数值，程序能得到正确结果。
出现不法数据呢？

实际软件应正确处理合法输入，还应**在输入有错时合理处置**。如果编译程序遇到不合法程序就垮台，还可能破坏操作系统，还有人愿意用它吗？

程序抵御不合法数据破坏的能力称为**强健性**。

函数readscores不会被错误数据破坏。循环条件中有数据项数控制，不会出现“输入缓冲区溢出”，数据过多只完成部分数据的统计。这里有缺陷：在数据没有用完的情况下也不声不响地产生了输出，却没有通知“工作结果可能不正确”。

统计函数检查了输入项数以防止除零的问题。

直方图函数检查项数，但由于采用优化实现方法，遇非法成绩时“`segs[(int)tb[i]/SEGLLEN]++;`”可能导致数组越界。问题是在数据入口未检查输入数据，非法数据可能混入，造成破坏。

问题与数据输入阶段有关。由于已将有关工作实现为函数，现在只需修改这个函数。

- 首先需要检查每个输入项，只将合法数据存入数组。
- 有关数据是否处理完的情况只能在循环结束后检查。

修改后的readscores如下：

```
int readscores(int limit, double tb[]) {  
    int i = 0;  
    double x;  
    //手工输入  
    while (i < limit && cin >> x)  
        if (0.0 <= x && x <= 100.0) {  
            ++i;  
            tb[i] = x;  
        } else  
            cout << "Data incorrect. Discard!" << endl;  
    .....
```

还可考虑其他改进。

6.6.3 统计源程序中的关键字

【例6-21】 写一个程序，统计在一个 C/C++ 源程序文件中各个 ANSI C 关键字出现的次数。

本例将用到学过的许多东西。这里没给出完整程序，重点是分析，提出了一些问题。

程序顶层框架设计：

打开文件

while (从文件中读得某个标识符s)

 if (s是某个关键字)

 相应计数器加1;

输出结果;

关闭文件;

打开文件

```
while (从文件中读得某个标识符s)
    if (s是某个关键字)
        相应计数器加1;
输出结果;
关闭文件;
```

让程序从文件流读取文件内容。

注意：源程序文件里面包含标识符、运算符、括号、标点符号和其它分隔符，标识符之间可能是由空格、制表符或回车符分隔的，也可能是由运算符、括号或标点符号分隔的。

无法从文件流中以整体读入的方法读入标识符，而必须使用文件流的**字符输入函数** `get()` **逐字符读入**，并设法提取出标识符。

统计关键字：关键字属于一个特殊的标识符小集合，只要对提取出的标识符判断是否属于这个小集合即可。

这提供了一种解决问题的线索：先打开一个源程序作为输入文件流，然后顺序读取该文件并识别其中的一个个标识符；得到一个标识符后判断它是否关键字；在遇到关键字时更新统计数据，最后输出统计数据并关闭文件。这套想法形成了一个解决方案。

注意，这是一个大选择，采纳它决定了后面工作中的许多东西。（另一可能方案是在读入字符的过程中直接统计关键字，例如，读到 f 后考查能否读到 or，成功时将for的计数值加 1。读者不妨沿这条思路继续考虑能否写出程序，有什么困难，如何克服。）



打开文件

```
while (从文件中读得某个标识符s)  
    if (s是某个关键字) 相应计数器加1;  
输出结果; 关闭文件;
```

打开文件是一项常规工作：定义一个全局性的输入文件流 inFile，用它打开一个原有的源程序文件。

```
ifstream inFile; // 全局的文件读入流  
char filename[56] = "hello.cpp"; //自行准备一个示例源程序  
//cout << "请输入文件名: "; //或在运行时输入源程序文件名  
//cin >> filename;  
inFile.open (filename); //打开文件输入流  
if (!inFile) { //如果打开文件失败，则 inFile 得到一个零值  
    cout << "错误：无法打开文件 " << filename << endl;  
    exit(1); // 显示错误信息并退出程序。  
}
```

⇒ 打开文件
while (从文件中读得某个标识符 s)
 if (s 是某个关键字) 相应计数器加1;
 输出结果; 关闭文件;

识别标识符是独立工作，定义为函数。为处理方便，函数不但识别标识符开始/结束，还应存储它。可用参数或外部数组。下面用数组参数。

函数应通告是否读到标识符。可返回**标识符长度**，0 表示文件处理完（考虑其他方式及优缺点）。

引进长度限制，函数原型：

```
int getIdent(int limit, char id[]);
```

主函数核心部分的框架：

```
while (getIdent(MAXLEN, s) > 0)
```

`getident` 可参考单词计数，画状态转换图：标识符开始和结束。
标识符：字母开头的字母数字序列，下划线作为字母。可用
标准库字符分类函数。

`getident` 应把标识符存入参数数组并记录字符个数。注意，C
程序对标识符长度没有限制。

可采用字符串形式：在识别的标识符后存一个空字符。从数
组内容可以知道标识符在哪里结束。

也可以不用字符串方式（应怎样处理？）。

细节：源程序可能有注释、字符常量、字符串常量等。应在查
寻下一标识符的过程中跳过。

标识符识别定义为函数，这些细节都隔离在函数内部，与函数
外的程序无关，可推后考虑。

打开文件

```
while (getident(MAXLEN, s) > 0)
```

⇒ if (s 是某个关键字) 相应计数器加1;
输出结果; 关闭文件;

必须以某种方式记录所有关键字。getident 把得到的标识符做成字符串，关键字做成字符串能方便比较。

ANSI C 共 32 个关键字，最长的关键字有 8 个字符。

可用一个 32×9 的两维字符数组，子数组按字符串形式存一个关键字。

```
char keywords[32][9] = { /* 定义和初始化 */  
    "areak", "case", "char", "const", "continue", "default", "do",  
    "double", "else", "enum", "extern", "float", "for", "goto",  
    "if", "int", "long", "register", "return", "short", "signed",  
    "sizeof", "static", "struct", "switch", "typedef", "union",  
    "unsigned", "void", "volatile", "while"  
};
```

```
打开文件
while (getident(MAXLEN, s) > 0)
    if (s是某个关键字) 相应计数器加1;
输出结果; 关闭文件;
```

将标识符字符串与数组中各字符串比较（用strcmp），可确定是否为关键字，并统计相应关键字的出现次数。

定义一个计数器数组存储各关键字出现次数，初始化为 0：

```
int cnt[32] = {0};
```

比较 s 是否为标识符并计数：

```
for (n = 0; n < 32; n++)
    if (strcmp(s, keywords[n]) == 0) {
        cnt[n]++;    break;
    }
```

打开文件

```
while (getident(MAXLEN, s) > 0)
```

```
    if (s是某个关键字) 相应计数器加1;  
输出结果; 关闭文件;
```

完成一种常见工作：确定一项数据是否在一组数据中，称为检索/查找。这里检索字符串，采用顺序比较方式。

计算中常需要做**检索**。数据可能很多，顺序检索太慢。人们提出了许多数据组织方法和检索方法。

（数据结构课有详细讨论）

打开文件

```
while (getident(MAXLEN, s) > 0)
    if (s是某个关键字) 相应计数器加1;
输出结果; 关闭文件;
```

getIdent 是关键！**请自己编程实现**。考虑一些有关的问题：

- 1) 用多大数组存标识符？C/C++ 对标识符长度没限制，而 getIdent 需要长度限制参数。遇超长标识符时能正确统计吗？何时出问题？出什么问题？
- 2) 标识符超长应采取什么处理原则以保证统计正确？如何合理处理？
- 3) C 关键字长度不超过8个字符。能否利用这个特性简化工作？（及早发现不是关键字；考虑把 getident 实现为“取得下一个长度不超过8的标识符”）。
- 4) 完善程序还须考虑 C 程序的特殊成分：注释、字符常量、字符串常量等。仔细分析这些问题，修改程序，使它能“立于不败之地”。